2

2

Case-Based Reasoning for Real-Time Problem Solving

Prepared by:

Kristian J. Hammond
Charles E. Martin
Christopher C. Owens

Department of Computer Science
The University of Chicago
1100 E 58th Street
Chicago, IL 60637

October, 1992

AFOSR-91-0112

Prepared for:

Air Force Office of Scientific Research
AFOSR-NM, Building 410
Bolling AFB, DC 20332-6448

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 1 Oct 1992	3. REPORT TYPE AND DATES COVERED FINAL, 1 Jan 91 - 30 June 92

4. TITLE AND SUBTITLE "Qualitative Real-Time Problem Solving" (U)	5. FUNDING NUMBERS 61102F 2304/A7
6. AUTHOR(S) Kristian Hammond, Christopher Owens, Charles Martin	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) The University of Chicago Department of Computer Science 1100 E. 58th Street, Ryerson Hall 152 Chicago, IL 60637	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Department of the Air Force Air Force Office of Scientific Research/NM Building 410 Bolling Air Force Base, DC 20332-6448	10. SPONSORING/MONITORING AGENCY REPORT NUMBER AFOSR-91-0112

11. SUPPLEMENTARY NOTES
AFOSR Program director: Dr. Abraham Waksman

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; Distribution unlimited.	12b. DISTRIBUTION CODE UL

13. ABSTRACT (Maximum 200 words)

This document summarizes the University of Chicago Artificial Intelligence laboratory's work on applying case-based methods to intelligent real-time problem solving. An approach to problem solving involving the storage, retrieval, adaptation, and re-use of successful strategies is outlined. The report describes work on an overall control architecture, on methodological issues in the development of representation vocabularies, and on memory organization for efficient storage and retrieval of cases. Work on six projects is described, including work on robot planning, the dynamic repair of transportation schedules, multi-agent cooperative planning, case-based design, and active perception. The result of this work is a model of planning and execution that handles the complexity and instability of a dynamic, real-time environment by making use of known plans stored in memory.

14. SUBJECT TERMS Artificial Intelligence, Real-time problem solving, Case-Based Reasoning, Memory organization.	15. NUMBER OF PAGES 28
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT SAR

A Objectives

The objective of our work on this grant was to address the complexity and dynamic nature of real-time problem solving through the application of case-based methods. Our work over the past eighteen months focused on three main ideas:

- The development of an **architecture** for encoding and storing representations of successful plans and problem-solving strategies for subsequent retrieval, modification, and application to novel problem-solving situations.
- The compilation of **vocabularies** for representing detectable failures and opportunities, and applicable repairs and optimizations, able to be stored in a dynamic memory and applied at run time.
- Embedding reactive systems in a higher-level architecture for planning and understanding.

These ideas found expression in six ongoing projects:

- DMAP/RAP, an integrated planning and control architecture being implemented on the University of Chicago Animate Agent robotics platform.
- RUNNER, a system for flexible plan execution in an everyday common-sense domain (a simulated robot working in a simulated kitchen),
- IOPS, an intelligent assistant for air transportation schedule repair,
- COHORT, a project that investigates coordinated, cooperative interaction between multiple agents,
- ROENTGEN, a computerized assistant for the design of radiation therapy plans, and
- POLYA, an intelligent geometry tutor.

B Status

The result of this work is a model of planning and execution that handles the complexity and instability of a dynamic, real-time environment by making use of known plans stored in memory. When a threat or opportunity is encountered, a system can use detectable features of the environment to retrieve an existing plan that is approximately appropriate to the current situation, and immediately begin executing the plan while simultaneously debugging and adapting it to fit the particulars of the current situation. The effect is a combination of the power of declarative plan structures combined with the flexibility of a reactive system.

The remainder of this section describes each of the five projects and discusses how it addresses the issues noted above.

B.1 DMAP/RAP

Martin's research has focused on embedding reactive systems in a higher-level architecture for planning and understanding. Reactive systems typically make use of pre-compiled knowledge to avoid computationally complex inference during execution. This makes their performance brittle in complex, changing environments, since the conditions at run time may differ from those under which knowledge compilation took place.

There are two general approaches to this brittleness: either the pre-compilation can take more information into account, or some means may be developed to dynamically re-program the reactive system under changing environmental conditions. The first approach can only delay the inevitable breakdown of the reactive system.

The second approach must necessarily involve a higher-level system capable of performing inference, with attendant computational complexities. Martin's research has therefore focused on a four-level architecture:

1. High-level agency software.
2. Low-level reactive planning software.
3. Control firmware.
4. Sensor and effector hardware.

This approach is being implemented on the University of Chicago Animate Agent robotic platform, which offers a strictly real-time test bed for these ideas. Intelligent real-time behavior depends upon:

1. Ability to reprogram the low-level reactive planning software using inferences about the nature of the environment.
2. Ability to manipulate a set of control firmware parameters without inference and in real time.
3. Ability to control the instantaneous response of hardware.

The control firmware level, though interesting in its own right, is not specifically a part of this research. The primary requirement is the enumeration of a set of control parameters that can be manipulated by the low-level reactive planning software.

The low-level reactive planning software is charged with the responsibility of achieving tasks assigned by the high-level agency software and making real-time changes to control routine parameters. These tasks are typically akin to the primitive actions of traditional AI planning research. The reactive planner must refine these tasks into more detail for the control firmware, it must allow for different refinements in different contexts, and it must cope with uncertain information and incorrect refinement choice. It must also keep the high-level agency software informed about significant changes in the world.

We are using the RAP reactive execution system for this low-level planning (Firby, 1989). The RAP model assumes that there is a prescribed method for carrying out every task in every situation. Each method is a set of actions that will accomplish the task in a given situation. Execution consists of choosing a task to work on, assessing the current situation, choosing an appropriate method, and carrying it out. The assumption of known methods makes task execution very much like hierarchical plan-expansion except that it takes place at execution time rather than being done in advance, and is done with reference to states of the world model that can be determined without inference. This assures real-time behavior.

The low-level reactive system is somewhat simpler than the original RAP system, since many of the memory issues are the responsibility of the high-level agent. In particular, the ability to connect sensor readings with previously- identified objects is now the responsibility of the high-level system.

B.1.1 Intelligent real-time behavior

Although the RAP system is capable of real-time response to its environment, it cannot cope with fundamental changes in that environment. This is the role of the high-level agent, and constitutes the primary focus of this research.

The fundamental idea is to create a high-level representation of the reactive system. This representation will include both the data structures *and algorithms* of the reactive system, and will be directly manipulable by the algorithms of the high-level agent. In particular, the high-level agent can build new reactive planning data structures to be subsequently incorporated into the low-level reactive system. Intelligent real-time response thus arises out of the interaction between the reactive system and the high-level agent:

- The reactive planning system provides continuous real-time response to the environment.
- The high-level agent responds to environmental changes by creating new reactive planning structures.

B.2 The high-level agent

The high-level agent is an extension of the Direct Memory Access Parsing (DMAP) architecture. Our goal is to use the DMAP memory structure to represent both planning and language understanding knowledge, and to achieve the planning and language understanding tasks using a single algorithm.

B.2.1 Uniform representation for planning

The key to success is the uniformity of representation. Here, for example, is a DMAP representation of the **move-arm** RAP.[1]

```
(DEFINE-NODE move-arm (ISA rap)
  (SLOTS (the-arm arm)
         (the-destination location)
         (the-success at-location
           (WHERE (the-object = the-arm)
                  (the-location =
                   the-destination)))))
```

Note that this definition of **move-arm** does not include context checks or method descriptions. Instead, the DMAP representation makes use of the

[1]The system uses a hierarchical representation of memory. For clarity, all keywords are in uppercase, and all relations are of the form "the-". All other symbols are memory node names. The appearance of "=" indicates a variable binding constraint.

hierarchical structure of memory to establish more specific instances of move-arm with the appropriate information.[2]

```
(DEFINE-NODE move-arm-internal (ISA move-arm)
  (SLOTS (the-destination location/truck-internal)
         (the-method prog2
           (SLOTS (the-1st fold-arm
                     (WHERE (the-arm = the-arm)))
                  (the-2nd primitive-move-arm
                     (WHERE (the-arm = the-arm)
                            (the-destination =
                             the-destination)))))))
```

This more specific version of move-arm represents the fact that the arm must first be folded before it can be moved inside the truck. The memory nodes fold-arm and primitive-move-arm are other raps, siblings of move-arm.

```
(DEFINE-NODE fold-arm (ISA rap) ...)

(DEFINE-NODE primitive-move-arm
  (ISA primitive-rap) ...)

(DEFINE-NODE primitive-rap (ISA rap) ...)
```

(Primitive raps invoke the true low-level RAP interpreter, usually resulting in subsequent changes to the robot control routines and possible sensor or effector actions.)

Method specifications The method specification for move-arm-internal was done by specifying constraints on the prog2 memory structure. This general structure represents the concept of taking two successive actions.

```
(DEFINE-NODE prog2 (ISA method)
  (SLOTS (the-1st rap)
         (the-2nd rap))
  (INDEX (the-1st) (the-2nd)))
```

[2]The following definition omits some aspects of move-arm, *i.e.*, the-arm and the-success. Constraints and relationships are inherited, so there is no need to respecify identical information.

The structure of prog2 is very simple, having as it does two substructures, both of which are raps. The fact :hat the first rap should come before the second rap is indicated by the presence of an index annotation to the DMAP node description.

The index annotation tells the DMAP system how it can recognize that the method has been achieved. The elements of the index indicate that the DMAP system must first recognize that "the-1st" has been achieved, and then recognize that "the-2nd" has been achieved. At that point, the DMAP system will be able to conclude, by virtue of the index annotation, that the prog2 has been achieved.

Context checks Note that move-arm-internal still does not specify a context check. This is because context checks as implemented in the RAP system are not necessary. The DMAP system relies on the structure of memory to perform context checks automatically. In this case, it is the specification of location/truck-internal as the-destination for move-arm-internal that results in the correct choice of method. The general transformation of RAP methods to DMAP memory structures is illustrated below.

RAP representation

```
(define-rap name
  (success success)

  (method                  ; method-1
    (context context-1)
    (task-net task-net-1))

  (method                  ; method-2
    (context context-2)
    (task-net task-net-2)))
```

DMAP representation

```
(DEFINE-NODE name (ISA rap)
  (SLOTS (the-success success)))

(DEFINE-NODE method-1 (ISA name)
  (SLOTS context-1
         (the-method task-net-1)))

(DEFINE-NODE method-2 (ISA name)
    (SLOTS context-2
           (the-method task-net-2)))
```

The role played by the context checks in the RAP system is now performed by specialization in the hierarchical memory of the DMAP system. The context checks of the RAP system are translated into slot specification in the DMAP system; for example, in move-arm-internal, the RAP context check (internal ?loc) becomes a more specific filler for the relation the-destination. Multiple methods become multiple sibling descendants of the more general rap structure. The DMAP system relies upon its algorithm for concept refinement within the memory hierarchy to perform method disambiguation.

Success tests The definition of move-arm specified a the-success relation, which was subsequently inherited by move-arm-internal. These relations correspond directly to the success checks of the original RAP system. At the highest level, every rap memory structure has a success test and a method specification.

```
(DEFINE-NODE rap (ISA mobject)
  (SLOTS (the-success state)
         (the-method method))
  (INDEX (the-success))
  (INDEX (the-method) (the-success)))
```

(The mobject node is the highest level of the memory hierarchy, and exists primarily as a reference point for the general specification of communicative acts.)

The general rap node serves as a reference point for two important indices. These represent how the successful execution of a rap can be recognized, as in the previously described annotation of method achievement. The RAP system specifies that execution is complete when the success test is satisfied; even when the methods are executed, the success test must still be checked upon method completion to determine successful execution.

Similarly, in the DMAP system these two indices represent rap success. There are two possibilities:

1. the success condition of the rap (the state that exists in the the-success relation) is recognized, or

2. achievement of the method associated with the `rap` (the `method` that exists in the `the-method` relation) is recognized, followed by recognition of the success condition as in (1).

In either case, the success condition must be successfully recognized in order to qualify as successful execution of the `rap`.

These two indices are inherited by all descendants in the hierarchy; for example, `move-arm-internal` can be recognized by its success condition (inherited from `move-arm`), or by its method followed by its success condition.

Taking action Although the previous discussion has been in terms of recognition, it is in attempting to recognize the successful achievement of methods that the DMAP system ends up taking action. One way to think about this is to take the prescriptive statement, "DMAP should now do a `move-arm-internal`." and recast it as a description of a state of affairs that DMAP should recognize, that is, "try to recognize the successful achievement of `move-arm-internal`."

How can that recognition be achieved? One way is by doing (recognizing) the achievement of `the-method`. Ultimately, the decomposition of tasks will be grounded in the attempt to recognize a primitive action such as `primitive-move-arm`. These recognition attempts result in calls to the lower-level RAP interpreter.

The second of the two indices associated with the general `rap` structure, then, is exactly that used to prompt task decomposition and the ultimate execution of primitive tasks in lower levels of the overall robotic platform.

of

B.2.2 Uniform representation for natural language

Once planning knowledge has been encoded in DMAP memory structures, it is straightforward to extend the natural language capacities in order to enable human users to interact with the system. For details about natural language understanding in DMAP, see (Martin, 1992). Here we will only demonstrate how references to planning structures can be recognized and disambiguated.

The same memory hierarchy used to represent planning knowledge is also used to represent the knowledge necessary for language understanding. This is necessary since it is exactly this planning knowledge which turns out to be essential to understanding many natural language references. For example, the communication "don't do that" can only be understood in reference to the current planning tasks of the robot.

Communicative actions The highest level of the language hierarchy in the DMAP system consists of the **mtrans** marker for communicative actions (Schank and Abelson, 1975).

```
(DEFINE-NODE mtrans (ISA action)
  (SLOTS (the-info mobject)))
```

This structure has many descendants, since it is the attachment point for the linguistic knowledge associated with memory structures. For example, the **move-arm** rap might be referred to using the following structure.

```
(DEFINE-NODE m/move-arm (ISA mtrans)
  (SLOTS (the-info move-arm))
  (INDEX move (the-info the-arm)
         to (the-info the-destination)))
```

In this example, the the-info *relation* has been specialized the the **move-arm** structure, and the structure as an index that gives a linguistic pattern that may refer to **move-arm**. The linguistic information is specified in a form much like that of a phrasal lexicon; a kind of template in which the variable elements are composed of memory relations.

In this case, the index begins with the word "move," followed by a memory reference to the **the-arm** of the **the-info** of the current structure. Chasing the relationships in the memory hierarchy, this reference results in the identification of the **arm** structure associated with the particular **move-arm**.

The situation is considerably more complex than we have indicated here, since the reference to the **arm** structure must also be by way of a communicative act. The definition of **m/move-arm** given above is therefore incorrect; a more correct version is the following:

```
(DEFINE-NODE m/move-arm (ISA mtrans)
  (SLOTS (the-info move-arm)
         (m-arm mtrans
           (WHERE (the-info = the-info the-arm)))
         (m-loc mtrans
           (WHERE (the-info =
                  the-info the-destination))))
  (INDEX move (m-arm) to (m-loc)))
```

Careful examination of this definition will reveal that it specified two sub-communicative acts as part of **m/move-arm**, each concerned with communicating a sub-structure of the **move-arm** structure. This allows the DMAP system to correctly resolve these sub-communicative acts with other **mtrans** structures, such as the following which specifies that the word "arm" may be used to refer to a robot arm.

```
(DEFINE-NODE m/arm (ISA mtrans)
  (SLOTS (the-info arm))
  (INDEX arm))
```

If there is more than one arm, of course, the DMAP algorithm will have to disambiguate between them.

The second specification of **m/move-arm** is correct, but is for many reasons not the best choice of representation. Efficiently handling issues such as syntax require more complex solutions, none of which are directly relevant to the subject of this paper. In fact, for simplicity we will specify **mtrans** structures as in the first (incorrect) definition of **m/move-arm**, but use the keyword "**MTRANS**" instead of "**INDEX**" as syntactic sugaring. In this paper we will not attempt to describe how syntactic concerns (subject-verb agreement, number agreement, etc.) are taken into account.

Referring to methods The kind of interactions we would like to have with our robotic assistants include instructions such as "fold your arm before moving it inside the bay." This might be used to *teach* the robot the **move-arm-internal** structure. The previous **rap** definitions impose the following abstraction structure:

```
(DEFINE-NODE rap (ISA mobject)
  (SLOTS (the-method method) ...) ...)

(DEFINE-NODE move-arm (ISA rap) ...)

(DEFINE-NODE move-arm-internal (ISA move-arm)
  (SLOTS (the-method prog2 ...) ...) ...)
```

In order to be instructed in **move-arm-internal**, the system must already know about the successive execution of two tasks. That is, it must know about **rap** structures whose methods are **prog2** structures:

```
(DEFINE-NODE rap/prog2 (ISA rap)
  (SLOTS (the-method prog2)))
```

This structure is more general than **move-arm-internal**, unrelated (but compatible) with **move-arm**, and more specific that **rap**. The addition of this structure turns the above abstraction structure into a lattice, since **move-arm-internal** will inherit from both **move-arm** and **rap/prog2**. Multiple inheritance is a critical aspect of the DMAP implementation.

The **rap/prog2** structure is the correct referent for the above human-robot interaction. We can define a communicative act to make that reference explicit in the DMAP memory.

```
(DEFINE-NODE m/rap/prog2 (ISA mtrans)
  (SLOTS (the-info rap/prog2))
  (MTRANS (the-info the-method the-1st)
          before
          (the-info the-method the-2nd)))
```

The instruction "fold your arm before moving it inside the bay" has thus been reduced to the phrasal pattern "*action-1* before *action-2*." Recognition of "fold your arm" and "moving it inside the bay" must be the responsibility of the **mtrans** structures associated with **fold-arm** and **primitive-move-arm**.

B.2.3 Algorithms

The DMAP system is designed to function within a large body of already-represented knowledge; inputs to the system are used to recognize these existing knowledge structures and modify them to represent what is unique about the current situation. At a high level, the algorithm may be expressed as two coroutines:

```
PROMOTE node:
If concept is a primitive operation
  then apply it and INTERPRET pending inputs; (•1)
  else gather indices to recognize node, (•2)
      for each index in indices,
          PROMOTE the first element of the index.


INTERPRET input:
If input is a promoted index element
  then adjust the node based on the input (•3)
      if there are no more elements in that index
```

```
then INTERPRET node associated with index;
else PROMOTE the next element of the index.
```

Three important elements are:

Primitive operations and input For the high-level agent, primitive operations are interactions with the lower-level RAP interpreter; many of these ultimately result in changes to the underlying control routines, while others communicate information back to the high-level agent.

This is critical for the success tests. The indices for `rap` both have a reference to `the-success` specified; since the concept referenced by this relation will always be a `state` structure, some means must exist for entering the INTERPRET coroutine with a `state` as input.

The communication is handled by primitives whose effect is to establish a monitor condition in the lower-level RAP interpreter. For example, if the success test of a hypothetical `rap` were that there was a blue object in front of the robot, the primitive would establish a monitor condition in the RAP interpreter that, in turn, sent a particular color histogram to the vision processor. When the vision processor signalled successful recognition of the histogram, the RAP interpreter would respond by resuming the INTERPRET coroutine with the appropriate `the-success state` structure as as input.

As another example, here are some of the definitions for `fold-arm`:

```
(DEFINE-NODE fold-arm (ISA rap)
  (SLOTS (the-arm arm)
         (the-success arm-folded
           (WHERE (the-arm = the-arm)))))

(DEFINE-NODE arm-folded (ISA state)
  (SLOTS (the-arm arm))
  (INDEX (MONITOR (folded-p the-arm))))
```

Where "`folded-p`" is a predicate in the world-model of the lower-level RAP interpreter. Note that no `the-method` relation appears in `fold-arm`; as with `move-arm`, they would appear as specializations.

Gathering indices This step updates indices that can refer to a node. By doing this dynamically, the state of promoted nodes is constantly changing to reflect the current state of the DMAP system. This is crucial for resolving ambiguity of reference.

As an example, consider the previous reference to part of the `m/rap/prog2` structure: "moving it inside the bay." "It" in this case refers to the arm, but how is that reference established? Recall the `move-arm` definition:

```
(DEFINE-NODE m/move-arm (ISA mtrans)
  (SLOTS (the-info move-arm))
  (MTRANS move
          (the-info the-arm)
          (the-info the-destination)))
```

After the recognition of the lexical item "move," the memory reference to the arm causes the promotion of the node representing that arm. The lexical item "it" is associated with the general communication of nouns; although there are many nouns, the recent promotion of the arm node disambiguates the lexical item.

Adjusting nodes Adjusting the associated node of an index implements a kind of "case-based reasoning" component of the system, in which specific prior instances are used to guide the current interpretation. This allows the system to make use of more specific knowledge as soon as possible in the interpretation process.

For example, an index used to recognize `move-arm` may be adjusted after the relation `the-destination` is found to refer to an internal location. Since the more specific `arm-move-internal` represents this more specific information, the system will adjust the node from `arm-move` to `arm-move-internal`. This means that *immediately*, the additional information represented at `arm-move-internal` is available. In this case, the knowledge that `arm-move-internal` requires that the arm be folded may prove to be relevant.

The more specific node also results in updates to the node promotions described above. This usually proves useful in disambiguation. Ease of disambiguation may result in further specialization of the associated node, and the two processes may feed back and forth in this manner many times during PROMOTE-INTERPRET interactions.

An important fourth element of the system deals with how to handle inputs that are contrary to the expectations established by promoted nodes. For details cf this process, see (Martin, 1991).

B.3 RUNNER

RUNNER is a system that uses plans in a commonsense domain (a simulated kitchen). As a research project it lies between traditional planning research and more recent work on situated and reactive systems, and draws from both areas.

Traditional planning systems construct action sequences (plans) that, when executed, will satisfy the goals given to the system. These systems need complete symbolic descriptions of the state of the world in order to construct the plans, and generally assume an exact model of the effects of actions. This sort of plan construction can be very computationally expensive, and is probably of limited utility in uncertain worlds.

More recent work (by Brooks, Chapman and Agre, and Rosenschein and Kaelbling) responds to the uncertainty of execution and the cost of plan construction by tying current action as directly as possible to current perception, maintaining little state over time. While they avoid the intractabilities of planning, these systems rely on delicate characterizations of agent-environment interactions by the designer, as well as on the assumption that everything needed to determine action is perceptually immediate. While robust to environmental uncertainty, these systems are brittle to patterns of interaction with the environment that have not been explicitly anticipated by the designer (*e.g.* problems such as looping, or becoming trapped in local maxima). Finally, since there is little declarative representation of knowledge in these architectures, it is difficult to see how to integrate anything but the most knowledge-poor kinds of learning within them. Reinforcement learning is the main technique that has been looked at within this framework (e.g. Brooks and Maes, Chapman and Kaelbling, Sutton), but there is good reason to believe that pure reinforcement learning suffers from combinatoric problems analogous to those that make classical planning intractable.

We draw from both traditions in the RUNNER project: We agree with the "situated" camp that it is not reasonable to assume that an action system has a complete symbolic model of the world before any action takes place, or that the behavior of agents can be well characterized without some characterization of the environments they are to inhabit. We also agree that traditional planning is too expensive to be invoked every time an agent wants to achieve a goal. But we argue for the utility of explicit symbolic representations of plans, largely because this sort of representation provides the information necessary for run-time recovery and repair, as well as for learning by incremental modification of those plans. These plans should be re-used as much as possible (thereby amortizing the cost of planning over time), so that routine action is computationally cheap. Nevertheless, the structure and causal dependencies of plans should be represented explicitly, so that problems can be diagnosed and repaired.

The central research issues in the RUNNER project are:

- Flexible plan execution, with learning from execution-time failures and unexpected opportunities.
- "Extrinsic" planning—recognizing in the midst of activity when it is appropriate to use a stored plan (as opposed to reasoning about the internals of the plan).
- The representation of plans to permit effective reuse in a changing environment.
- Stabilization of the environment by the enforcement of policies.

Of these, probably only the last requires explanation: the idea here is that plans to be reused need certain preconditions to be re-established, and also that the problem of plan reuse is simplified by ensuring that certain of these preconditions are "always true". This is an abstract phrasing of a simple idea: for example, the fact that most people make sure that they always have clean drinking glasses in a particular kitchen cupboard probably makes it much easier for them to decide what to to do when they decide they want to have a drink of water. This sort of "stabilization" of the immediate environment trades off with the need for flexible plan representation—the more standardized an environment is with respect to the preconditions of a plan, the less flexibility is needed in the plan's execution.

The RUNNER program interacts with a simulator, which provides for it the illusion of inhabiting a conventional kitchen. Although a simulation, it is a fine-grained one—the RUNNER program controls a "head" with a visual focus and a limited field of view, and "hands" which must be directed by visual attention to manipulate objects in the field of view. We hope to make the simulator available to other research labs soon.

In this simulated world, RUNNER must perform simple tasks like making coffee and breakfast cereal. RUNNER starts out with a small set of hand-coded plans. Its job is to use these plans (and interleave their actions when necessary) to actually achieve the goals in its world. These plans represent the causal dependencies of their different parts, but do not completely determine the action of the agent: ordering of steps can depend on discovered opportunities, steps from different plans can be interleaved, and the appropriate times for particular plans must be recognized on the basis of perceptual cues.

Achievements in the past eighteen months include:

- Development and extension of RUNNER's simulated world. Some features of this simulator are:

    1. A fairly sophisticated "physics", in which unsupported objects fall, struck objects acquire momentum and move independently, and objects approach the temperature of the surrounding environment.
    2. A simulation of visual search, which enables RUNNER's agent to look initially for objects on the basis of color or texture, and then focus on candidate objects to see if they are of the right type.
    3. Complete decoupling of the simulator and the AI program; the two programs can now run on different machines, and communicate over UNIX sockets. Multiple AI programs (not just RUNNER) in the Chicago AI lab can "attach" to the simulator and inhabit the same simulated world.

- Successful use of plans that result in action sequences of four to five hundred steps.

- Successful "multiple day" examples, where the program uses the same plan representation to perform a task even though its own activity has changed the circumstances of the relevant objects (*e.g.* an object which was taken from a cabinet is now found left on a counter on the next run of the plan).

- Implementation of a simple form of speedup learning, where the program learns to choose particular methods for subgoals based on failure rate and time spent (e.g. some objects are more easily found by color and others by texture).

- An unexpected example of opportunism: due to a quirk in the simulator, some ground coffee was left over in the simulated coffee grinder from the previous day's coffee making. While visually searching for coffee to grind, RUNNER noticed the coffee grinder and decided to use the coffee that was there and abandon the search it was engaged in. This pointed us to a bug in the simulator to fix, but in terms of the world RUNNER found itself in it was opportunistic behavior that we did not foresee.

B.4 IOPS

We have identified *airline irregular operations management* as a task that embodies the key features of dynamic problem-solving in general. Although this domain involves decision-making over a time span of minutes to hours rather than the seconds that characterize some other real-time problem-solving domains, the key properties of incomplete information, time-criticality, the asynchronous arrival of new information, and the computational intractability of first-order solutions nevertheless apply. Our approach to this problem is to gain computational leverage by noticing and exploiting the similarity between new situations and stereotypical, recurring ones.

B.4.1 The irregular operations problem

An operations schedule for a large airline is a complex structure including flight segments, equipment and crew routings, and routine maintenance operations. It is built to satisfy a large number of independent constraints, and it contains massive interdependencies between elements.

The static schedule (the schedule the airline would fly under ideal circumstances) is established in advance, and is carefully optimized using quantitative techniques developed by the operations research community. But unpredictable events inevitably occur, forcing the actual schedule to deviate from the static schedule. Weather can close an airport or reduce throughput, aircraft can require unscheduled maintenance, and crew can be unavailable. Because of the interdependencies in an airline schedule, even a single disruption and the consequent attempts at recovery typically involve widespread and long-lasting downstream effects. The search space of possible recoveries to a schedule disruption is enormous.

Airlines employ operations controllers, whose job it is to mitigate the effects of operations disruptions. They can cancel or reroute flights, add intermediate stops to pick up stranded passengers, move empty aircraft around the system, substitute equipment or crew to cover shortfalls. Their main goals are to minimize both passenger inconvenience and the cost of implementing the repair, while accounting for crew work rules, aircraft maintenance schedules, and other factors. An additional goal is to minimize the overall complexity of a repair. Operations controllers operate under time constraints due to the inherent time constraints of a schedule and the long lead times necessary to implement many plan repairs.

B.4.2 Encoding and applying cases

Our main scientific goal in this area is to develop a theory of failure, detection, and repair in the scheduling domain. The main result of the project will be an "intelligent assistant" for schedule controllers: a system whose job it is to notice and exploit the similarities between new situations and stereotypical, recurring failures that have been successfully solved in the past. Reusing old solutions is expected to be faster and more robust than generating new solutions from first principles.

To this end, we have established a cooperative relationship with United Airlines that enables us to study and analyze the actions of skilled, experienced flight controllers as they solve real problems. Based upon our observations of tne controllers, we have been characterizing, and encoding a library of typical, recurring failures and appropriate repair strategies. This library forms the core of IOPS (Interactive Operations Planning System), an intelligent planner's assistant now under development.

Our initial study has suggested to us that controllers build and use sophisticated, high-level repairs from a small number of primitive operators. The primitives form the basic representation vocabulary used to describe actions, and it is anticipated that the list will be stable over time. The higher-level strategies, on the other hand, are more dynamic, and one of our tasks is to model the acquisition of new high-level strategies.

Typical primitive operators represent concrete actions like:

- Cancel a segment
- Delay a segment
- Divert a flight to a different airport
- Substitute one aircraft for another
- Substitute one crew for another
- Ferry an empty aircraft from one airport to another

Higher-level strategies, on the other hand, may involve both primary actions and secondary actions designed to mitigate the side-effects of the primary actions. Or, they might involve a series of steps taken to defer the impact of a problem, in the expectation that an opportunistic solution may present itself in the intervening time. Other high-level strategies include geographically localizing the impact of a problem or, conversely, diluting the impact of a problem by spreading a minor delay across several geographic points.

A longer-range goal for IOPS is that, in having a human user interact with a planning tool, we have an opportunity to record information about plan accessing strategies, modification techniques and typical failures that can, in turn, become the heuristics used by a more autonomous system. A system that observed human schedulers in action and recorded their responses to specific planning problems, and which indexed those responses in memory using the functional criteria discussed above, would become a powerful expert assistant — an assistant with a good memory for what worked and what didn't in the past.

Our ongoing work in representation is aimed at encapsulating the high-level strategies with recognition criteria to enable IOPS to determine the applicability of strategies to new situations, and at testing a system's ability to detect the applicability of these high-level strategies given real operating data. We are now able to capture schedule and operations data from the airline; we are currently developing the representational and algorithmic capability to apply the high-level strategies and associated recognition criteria to this real data.

B.4.3 Enabling quantitative solutions

Reasoning about transportation scheduling is a task for which a number of well-understood quantitative techniques have been developed by the Operations Research community. The computational costs of these techniques, however, become large when applied to large schedules with complex inter-dependencies. One of our scientific goals has been to apply the qualitative solutions described above (pairing detection criteria with repair strategies for recurring failures) to the task of heuristically limiting the complexity of standard quantitative solutions for scheduling problems.

One example of such an interaction is to use the qualitative solution to generate a set of choices, and then to use quantitative optimization techniques to choose among the choices. For example, if the system selects *add* *an unscheduled intermediate stop* as a repair to accommodate some otherwise stranded passengers, the choice having been made based upon the qualitative features of the situation, standard optimization models can be used to select which among the candidate flights should make the stop to pick up the passengers.

The challenge here is to use the qualitative repair strategies to obtain some locality — to avoid the need to run the quantitative techniques over the entire network. Each qualitative repair strategy must include, as part of its representation, likely pieces of the overall schedule to consider when running the network optimization code. Although the downstream dependencies of a single failure or repair potentially involve the entire network, and although a guaranteed optimal solution would require considering the entire network, our plan is to test the ability of pre-compiled repair strategies to provide an acceptable kind of satisficing, rather than optimizing, solution to the scheduling problems.

B.4.4 Status

Our primary effort over the past eighteen months has been in gathering data about the actions of skilled controllers, and in developing the computational capability to represent and reason about an object as large and complex as an airline operating schedule. We have established our own machine-readable representation of the airline's complete operating schedule, including the ability to represent live data concerning current operations. This undertaking consumed a significant portion of the first year's effort on the project, and we now have a capability for examining the consequences and costs of a schedule failure, and for predicting and reasoning about the downstream effects of repairs. We have in place a rudimentary graphical interaction capability, which we plan to extend to be the primary mechanism for interaction between schedulers and the system.

Our most encouraging result to date has been the apparent convergence of a small set of repair strategies that, suitably modified, appear to cover a large variety of failures. Our ongoing work is directed at testing the validity of this appearance.

B.5 Roentgen

ROENTGEN is a computerized assistant which aids in the design of radiation therapy plans. Such plans use beams of high-energy photons to treat cancer.

The success of radiation therapy depends partly on the skill of the therapy plan designer. One of the motivations for the project has been to provide a concrete tool for this necessary task.

In order to develop this tool, I have been concerned primarily with three issues:

- **Apprentice-Assistant-Advisor Systems.** How to design interactive systems whose abilities grow through experience from being apprentices, to helpful assistants, to being skilled problem solvers.
- **No Perfect Solution.** How to solve a problem when none of the possible solutions satisfies all the desired goals.
- **Symbolic Representation in Graphical Domains.** How to reason symbolically about problems that are specified graphically.

The basic approach behind ROENTGEN has been to take standard case-based reasoning methods and investigate how they must be adapted to cope with these issues.

ROENTGEN utilizes its memories of past cases, failures, and failure analysis episodes to perform its work. It has a *Retriever* to find already-solved treatment problems similar to the patient for whom a plan is being developed. The *Adapter* tailors the retrieved plan to the current patient. The *Evaluator* analyzes the results of simulating the plan on the current patient and compares what it finds with what it expected to find. The *Evaluator*'s expectations are based on past experience. If it finds unexpected faults, the *Explainer* produces a causal explanation for the faults. The explanation links features present in the patient with the faults and with parameters of the plan. Armed with an explanation, the *Repairer* adjusts the plan's parameters so it no longer produces the faults. Finally, the *Storer* places good plans in case memory where they may be found and used in the future.

ROENTGEN begins its practical life without an internal representation of the physics of its domain. Everything it needs to know will be explained to it by the human expert when the knowledge becomes necessary. For instance, ROENTGEN has no prior knowledge about what features to use when judging how similar patients are. As the system participates in the design of plans, the human expert defines the important features for ROENTGEN. Similarly, the expert defines undesirable conditions in the results of plan simulation as they occur. The system will pay attention to these features and notice these faults in its future work.

In order for the human expert to be able to define features, faults, repairs, and so on, she must have vocabularies of primitive terms which can be combined to produce the needed concepts. An important part of the work on the ROENTGEN project has been to develop these vocabularies of primitives. They provide the link between the graphical representation of the problem and ROENTGEN's needs as a case-based reasoner.

ROENTGEN determines how well a plan performs by comparing the results it achieves with the results of similar plans applied to similar patients. If the current results are at least as good as past ones, ROENTGEN decides the plan is performing well. By using past experience in this way, ROENTGEN can judge when it has a good—but, less than perfect—plan.

While ROENTGEN is proceeding well now, eighteen months ago the project was in some difficulty. The main problem was that the features used by the system, at that time, were not tightly connected to the physics of radiation therapy. For example, the feature TARGET-LOCATION described the location of the tumor to be treated using the coordinates of a rough grid. Such a description was not precise enough to be causally significant. Since the features ROENTGEN was using were not causally significant, they could not be used successfully for retrieval, fault analysis, and repair.

So, the first achievement of the past eighteen months has been to discover the correct feature vocabulary for the domain. Once, the project was on the right track in this respect, progress was made in other areas as well. The main achievements for this period are:

- The primitive vocabularies for patient features, faults in plan application, plan repairs have been established.
- The basic structure of ROENTGEN—the modules which make it up, their function, and the knowledge they need—has been defined.
- The method of using past results to judge current plan results was developed.
- Computer code using the new ideas about feature vocabulary in retrieval, evaluation, explanation, and repair has been largely written.
- A better understanding of the interaction between the human and the system, as the system accumulates knowledge, has been achieved.
- Papers describing these advances have been presented at CBRW91 and SCAMC92.

B.6 Polya

In POLYA we are building the core of an intelligent geometry tutor. This core should be able to solve geometry problems in a way which would make sense to a student and which the student could profitably imitate. We believe that a relatively small fraction of expertise in geometry comes from the traditional formal geometry knowledge. Expertise comes from enhanced visual skills and the ability to recognize common and meaningful configurations in the diagram.

Furthermore, the core system should be able to introspect about its own problem-solving activity and explain it to the student.

We are also expanding the traditional notion of geometry knowledge. This new notion emphasizes the diagram: what are the important features, how to detect those features efficiently, and how those features are likely to tie in to the proof.

Our progress towards a functional goal is shown by a working program, POLYA, which can currently solve simple triangle congruence proofs. Examples of problems which POLYA can solve are given in figure X.

The significant results of our research so far lie not in what problems POLYA can solve, but in how POLYA solves them. As stated before, our approach emphasizes the use of the diagram as a source of features for indexing into a memory of geometry proof plans. However, for our system to be effective as a tutor, it must acknowledge the limited ability of novices to parse the diagram at once, and their need to learn what to look for and what to ignore.

Therefore, we have made explicit POLYA's knowledge of how to use the diagram effectively. That knowledge includes a vocabulary of detectable visual features, visual actions for detecting those features, and purposeful sequences of visual actions called geometry *scripts*. The features, actions, and scripts which POLYA uses to solve geometry problems form a new theory of geometry expertise, and thus represent the significant theoretical results of our research.

The past 9 months have seen the most rapid progress on POLYA. Late January, 1992: invented geometry scripts for visual search and for writing pieces of proofs, and an initial set of visual actions. A preliminary paper describing scripts and how they could support geometry problem-solving was accepted to the AAAI Spring Symposium on Reasoning with Diagrammatic Representations.

By February, we had implemented an interpreter of geometry scripts and had written scripts to solve a few examples. A paper describing the working system was accepted to the Cognitive Science conference.

The current system defines 50 actions which are used in 20 scripts, and can solve 12 examples.

B.7 Cohort

Our concern in the COHORT project is to investigate planning and situated action in multiagent worlds. Most research in planning for multiagent environments has involved distributed problem solvers. Agents in these centralized systems pursue a common goal and try to achieve it through cooperation with the others. Since these agents may have different capabilities and incomplete, local knowledge about the state of the global system, an important issue in this type of approach is that of designing agents and communication protocols that allow them to cooperate in coherently achieving the overall goal. Our approach in the COHORT project is completely different. We believe that to achieve coherent overall multi-agent behavior, we have to extend our theories of planning and acting, (or *agency*) to account for *social agency*. A social agent is an autonomous agent which plans and acts in a multiagent world, and which can smartly interact with others and learn from these interactions. In particular, such an agent should be able to decide whether, when, why and how is appropriate to cooperate with others.

In the COHORT project, we have two main objectives. The first is to develop a vocabulary that will represent our theory of social agency. The second is to implement a multiagent system composed of autonomous interacting agents which use this vocabulary to reason, act and learn from their interactive experiences in this world. Not only must this vocabulary allow that agents take situated action in their worlds, but it must also allow them to learn over time how to better act and interact. In some cases, this learning may give rise to multi-agent routines that serve to improve the agents' behavior.

Over the past eighteen months we have been laying the groundwork to pursue these two objectives. In particular,

- we developed a knowledge representation scheme that builds upon earlier work in *opportunistic memory* and *agency* done at the Univ. of Chicago. This scheme serves to represent the agent's knowledge in a useful way so that it can be retrieved at the appropriate times during the agent's involvement in its activities. The agent's knowledge includes planning knowledge that allows it to pursue individual and social goals. Additionally, it contains social knowledge that allows it to engage in common and recurrent types of inter-agent interactions in its multiagent community,

- we developed the beginnings of a vocabulary of interactions for these agents. This vocabulary of interactions seeks to characterize what is important in a particular inter-agent interaction so that agents can use this knowledge to learn from their experiences.
- we implemented a few related examples of inter-agent interactions in in a simulated maintenance world. These examples involve two agents that pursue cleaning and furniture moving tasks. The agents are capable of individually and jointly engage in concrete activities.

We expect to continue improving our vocabulary of interactions as we experiment with our multiagent system. We are currently addressing the issue of building learning capabilities into our agents.

C Publications

Tim Converse and Kristian Hammond. Learning to Satisfy Conjunctive Goals. In *The Proceedings of the 1992 Conference on Machine Learning.*

Kristian J. Hammond and Colleen Seifert. A Vocabulary for Indexing Plan Interactions and Repairs. In *The Proceedings of the 1992 Meeting of the Cognitive Science Society.*

Patricia Goldweic and Kristian J. Hammond. Multi-Agent Interaction. In *The Proceedings of the 1992 Meeting of the Cognitive Science Society.*

Tim Converse and Kristian J. Hammond. Preconditions and Appropriateness Conditions. In *The Proceedings of the 1992 Meeting of the Cognitive Science Society.*

Kristian J. Hammond, Timothy Converse, Mitchell Marks, and Colleen Siefert. Opportunism and Learning. In *The Journal of Machine Learning.* In press.

Kristian J. Hammond, Tim Converse, and Mitchell Marks. Towards a Model of Agency. In *Machine Learning Methods for Planning,* ed. by Steve Minton and Pat Langley. In press.

Charles E. Martin and R. James Firby. An Overview of the Dynamic Predictive Memory Architecture for Robotic Assistants. In *Proceedings of the Third Conference on Cooperative Intelligent Robotics in Space*

Christopher C. Owens. A Functional Taxonomy of Abstract Plan Failures. in *Proceedings of the 1991 Conference of the Cognitive Science Society.*

Christopher C. Owens. Integrating Feature Extraction and Memory Search. in *The Journal of Machine Learning.* In press.

Christopher C. Owens. Problem-Solving Stereotypes for an Intelligent Assistant. In *Proceedings of the 1992 Conference of the Cognitive Science Society.*

D Personnel

The principal investigators on this grant were Kristian Hammond, Christopher Owens, and Charles Martin, all faculty members at the University of Chicago Department of Computer Science. Martin is collaborating with R. James Firby, another faculty member, on the DMAP/RAP project. Owens is developing the IOPS system. Hammond is supervising the remaining projects.

In addition, the following PhD. students were active on the projects associated with our work on this grant:

- Jeffrey Berger (the ROENTGEN project.)
- Timothy Converse (the RUNNER project.)
- Patricia Goldweic (the COHORT project.)
- Thomas McDougal (the POLYA project.)

E Interactions

E.1 Conferences and workshops

In addition to the presentations listed under "Publications", members of the group made the following conference and workshop presentations.

Owens and Martin made presentations at the 1991 AFOSR-sponsored "Intelligent Real-Time Problem-Solving" workshop held at Wright-Patterson AFB.

In May, 1991, Owens attended an NSF invitational workshop on AI Methodology, at Northampton, Mass.

In June 1991 Jim Firby and Charles Martin were on the organizing committee for a workshop on "Learning Reaction Strategies," and Christopher

Owens was on the organizing committee for a workshop on "Information Retrieval and Machine Learning, both held at the Eighth International Machine Learning conference.

At the 1991 *AAAI Fall Symposium on Knowledge and Action at the Social and Organizational Levels,* Goldweic discussed "Multi-agent Interaction: Dynamics of Cooperation"

At the 1992 *Midwest Artificial Intelligence and Cognitive Science Conference,* Goldweic presented her work on the COHORT project, John Borse (a Masters' student working with Owens) presented work in progress on the IOPS system, and McDougal presented an overview of the POLYA project.

Hammond presented an invited planary talk at the 1992 AAAI conference.

E.2 Consultative and advisory functions

Our work on this grant is synergetic with our work on the DARPA Planning Initiative In particular, our work on failure and repair in the airline scheduling problem squarely addresses one of the Planning Initiative's task areas. We expect that the catalogue of failures and repair strategies developed under this contract will be directly applicable to our work developing a planner's assistant for the PI.

In March, Owens attended Neal Glassman's AFOSR-sponsored workshop at Scott AFB to learn about USTRANSCOM's approach to transport planning and scheduling. USTRANSCOM has a problem apparently exactly analogous to the airline schedule repair problem; we have established a contact (LtCol Joseph Litko) at USTRANSCOM who has referred us other sources of information about USTRANSCOM's planning and scheduling activities. We expect to pursue these contacts further this Fall.

F Discoveries, inventions, patent disclosures

Not applicable.

References

R.J. Firby (1989), *Adaptive Execution in Complex, Dynamic Worlds.* Ph.D. dissertation, Yale University.

Charles E. Martin and R. James Firby (1992). An Overview of the Dynamic Predictive Memory Architecture for Robotic Assistants. In *Proceedings of the Third Conference on Cooperative Intelligent Robotics in Space*, Boston, MA.

Charles E. Martin and R. James Firby (1991). Generating Natural Language Expectations from a Reactive Execution System. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, Chicago, IL.

Charles E. Martin and R. James Firby (1991). An Integrated Architecture for Planning and Learning. In *ACM SIGART Bulletin* **2**(4), 125–129.

Charles E. Martin and R. James Firby (1991). Advising a Reactive Planner. In *First Annual Workshop on Planning and Learning*, Stanford, CA.